Decoupling database keys from caBIG ids

Database keys and caBIG ids should not depend on one another, because the requirements of maintaining an underlying database and the requirements of providing proper grid ids are different and may conflict. Allowing them to vary independently avoids tendencies to satisfy one set of requirements at the expense of the other. This would mean that grid data services would be required to add columns to database tables to store the caBIG id separately and alongside the database key. Alternatively, the grid service could generate the grid id on the fly if there were some reliable way of doing so (e.g. mapping an accession number to a caBIG id; note that this still keeps database primary keys separate from caBIG ids).

Resolvable ids

In order for an id to be resolvable, some part of the id must enable discovery of the originating data service. LSIDs provide the domain name of the originating authority, but because the grid id is permanent, this complicates resolution of the service if the service is moved to another host. Instead of a DNS domain name, I suggest that part of the caBIG id be a unique service name, which can be resolved via a caBIG (LDAP) directory service, parallel and integrated with the distributed vocabulary service. In this way service names become just another type of caBIG vocabulary. Because this naming scheme is decoupled from standard host domain names, we can allow services to be hosted at arbitrary and modifiable domain names, so that services could easily be moved to other servers. Because resolvability extends to the individual services themselves, the services offered by a given host could be later split among several hosts. Alternatively, one could group many retired services onto a single server.

One might conceive of the id this way. Use something similar to the format of an LSID (say we call it a cabigid for lack of a better name):

urn:cabigid:service name:object id:version id

I have coalesced the LSID authority and namespace into a combined "service name". This service name would be the name of a node in a (LDAP) directory information tree, referring to a particular service on the grid.

It is important that the service names be organized in a directory tree structure, allowing for efficient resolution and delegation with LDAP (a la DNS). This would integrate nicely with a general vocabulary services structure based on LDAP, and would provide for flexibly manageable and highly performant mappings of service vocabulary terms to actual services on the grid. For example, if a semantic renaming of services were necessary (not just moving services to another host, but reorganizing the directory tree structure), it would be easy to place aliases for the correct service name at the old LDAP

tree nodes (just as is done in DNS). Also, different servers could be authoritative for different "zones" of the directory tree, with other servers delegating to them as approrpiate, allowing for distributed hosting and governance of the directory (again, a la DNS). A given directory server can cache lookups from another authoritative server, improving overall performance and reducing bandwidth. Finally, one can easily configure "shadow servers", which mirror a primary authoritative server for a particular zone of the tree, and receive updates from the primary, relieving the load on the primary server. If you already know about DNS, I'm boring you, but I was happy to discover that these important features are built in to LDAP as well.

The node in the directory tree corresponding to a given service name would contain resource records (in DNS parlance) or attributes (in LDAP terms) identifying the domain name, port, WSDL URL, etc. (or domain names, ports and URLs) of the service (replicas of a service) corresponding to the term. Other attributes could be imagined, but I'll leave that to later discussion.

The service name could take either the simpler DNS form of a set of names separated by periods (genbank.genes.data) and be automatically converted via a standard algorithm to the LDAP distinguished name for the node, or could just use the more flexible but lengthier LDAP distinguished name form directly (attribute=value,attribute=value,...), e.g., sc=data,sc=genes,sn=genbank (I'm inventing, for the sake of example only, possible new attributes in an LDAP schema like sc (service component) and sn (service name)).

Naming conventions for these service vocabulary terms would have to be invented. At the risk of being redundant, note that because these service names do not have to correspond to DNS names, they can be organized according to a tree structure most appropriate for use as namespaces for services and grid ids. This should resolve the concerns about id semantic opacity that I read about with respect to LSIDs (at least with respect to the authority and namespace), unless I don't understand all of the concerns there. That is, there is no need for opacity because the naming scheme frees you to name your services appropriately, and allows you to correct your naming scheme (via aliasing) as necessary.

There could also be multiple types of service naming trees, e.g. classifying by function, by hosting institution, by service level (?), allowing users to look up services along a variety of axes. Alternatively, one could do LDAP queries on the attributes of service nodes, so that one would not need to actually construct different trees. There are probably performance implications of choosing one or the other of those options. Also, only one of the naming trees should be used for providing the service name component of the grid id.

Importantly, this could be very simple in the beginning (one host), and get more detailed, redundant, etc. as needed.

Provisional testing period

I think there should be a period of caBIG when the grid ids handed out are provisional, so that we have a chance to learn about the problems of our identifier architecture from a working system.

Publishing to the grid means publishing forever

As someone else said: When an identifier is returned to a client, that identifier should not disappear in the future. A request for that identifier's object should return a response, even if that response is something like "Retired".

However one might want to avoid "Retired" and always keep a copy of old data. A use case follows:

Say the software providing a particular caBIG analytical service is found to contain a bug, and say papers have been published based on conclusions reached through the analysis offered by that service. One would want to check and see if the conclusions made change with the corrected algorithm. In order to check this reliably, one would want to check the changed algorithm against the exact same data set used in the original analysis. Being able to access previous, potentially incorrect versions of the data objects would be important in this case. On the other hand, maybe if the data were updated, it would be ok to just run against the fixed data.